GigaVoxels-GigaSpace Engine

Production

Profiling & Optimizations

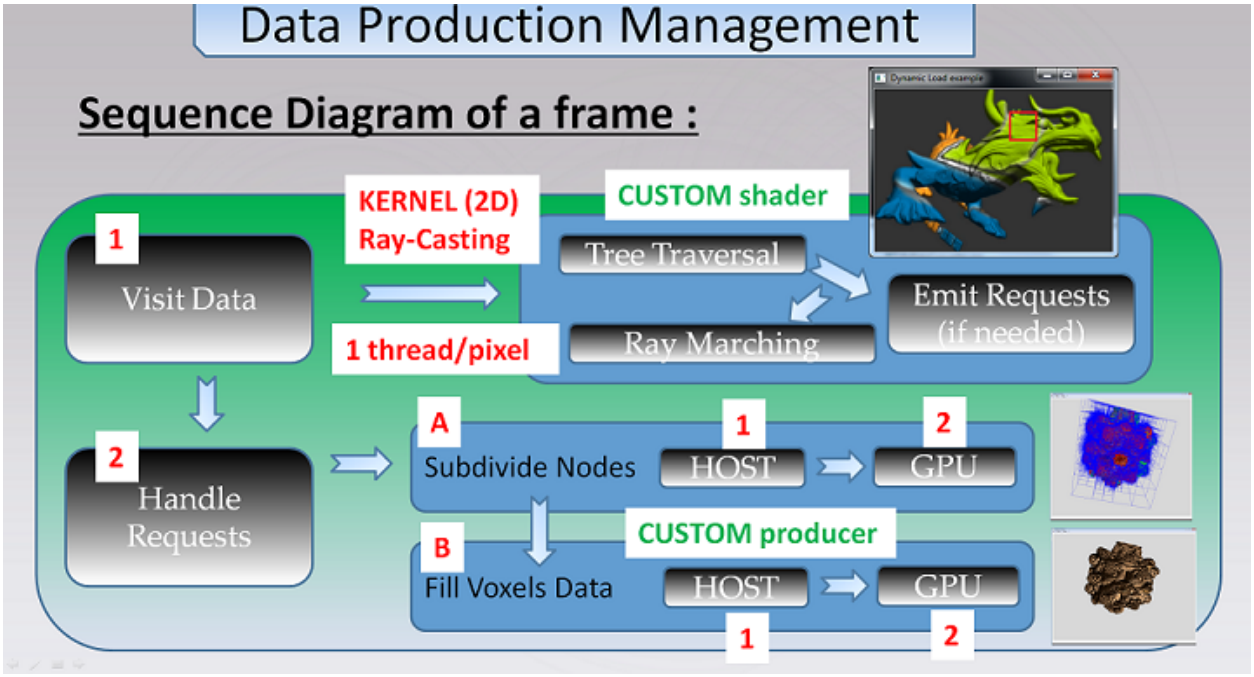
1. Introduction

a. Data Production Management

The Data Production Management is responsible for the production of nodes first and then bricks:

- process node requests (first CPU, then GPU stage)
- process bricks requests (first CPU, then GPU stage)

This is where users have to define their own CPU and GPU producers :



Node subdivision

A node subdivision request is raised when user need more resolution, i.e. a data refinement. For this, GigaVoxels launches a CUDA kernel to subdivide all requested nodes. For each node, the goal is to say what will lies in its children (at a finer level of resolution). USER has to define what we call an ORACLE, i.e. a utility function used to predict what lies in children nodes.

- KERNEL : 1 bloc per node and 1 thread/child_node -- Each node has to say what's inside each of its child
- INPUT : localization info of current node (LOD depth and spatial index pos)
- INPUT : address in "node cache" where to write new child nodes
- Test if it is in a sphere (analytically)
- Write nodes in cache

2. Profiling

a. Configuring the NSight Environment

Check the report “Graphics Library Profiling and Optimizations”

b. Example : Simple Sphere demo

Here’s the timeline of the GL calls during 1 frame :

TO DO

...

3. Optimizations

a. Kernel “internal synchronizations”

Actually, the GigaVoxels-GigaSpace Engine provide one kernel to produce either nodes or bricks. The use of “C++ templates” enables to provide generic behaviour for the two. However, due to different approach in their associated “launch configuration” (i.e. grids and blocks of threads).

Each kernel requires to, first, retrieve common information. A “synchronization” barrier is used to ask only one thread to get these data.
During “node production”, we process 1 node subdivision by 1 block of thread. Say we work with octree (i.e. 8 child to produce), we should launch blocks of 8 threads. But, due to hardware constrainits, the CUDA device main scheduler organized threads in groups of warps of 32 threads each. In a warp, all threads are synchronized. So, there is no need do ask for a costly “barrier synchronization” in these kind or kernel.

The idea is tyo be able to retrieve “Warp Size” at program initialization and launch different kernels based the use of synchronization or not, asa “dynamic dispatcher”. C++ template could be a nice choice for this.

Even if it is said to not write programs based on warp size, it’s a very common optimization in kernels such as parallel prefix sum.

TO DO : add graphs + timelines
…

PREVIEW => detail and explain :

<u>Usual Nodes Production (to produce all required nodes given default view parameters)</u>

1/. First test

Drag a column header and drop it here to group by that column

				Duration (µs)				Grid Dimensions	Block Dimensions	Dynamic Shared Mem/Block	Stream ID	Occupancy	Registers Per Thread	Cache Configuration Executed	Shared Memory Configuration Executed
1	52	73	95	4.992	4.5	4.5	Gv	{1, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
2	64	12	96	4.672	32	32	Gv	{8, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
3	75	18	96	4.576	17	17	Gv	{44, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
4	81	21	96	4.448	47	47	Gv	{12, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
5	87	24	96	7.681	60	60	Gv	{172, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
6	94	27	97	4.513	22	22	Gv	{56, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
7	10	30	97	18.946	1,7	1,7	Gv	{636, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
8	10	33	97	8.833	68	68	Gv	{228, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE

2/. Second test

Drag a column header and drop it here to group by that column

				Duration (µs)				Grid Dimensions	Block Dimensions	Dynamic Shared Mem/Block	Stream ID	Occupancy	Registers Per Thread	Cache Configuration Executed	Shared Memory Configuration Executed
1	52	73	1,1	4.865	4,4	4,4	Gv	{1, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
2	64	12	1,1	4.672	32	32	Gv	{8, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
3	75	18	1,1	4.544	17	17	Gv	{43, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
4	82	21	1,1	4.513	16	16	Gv	{4, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
5	88	24	1,1	7.489	61	61	Gv	{181, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
6	95	27	1,1	4.385	46	46	Gv	{12, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
7	10	31	1,1	20.418	1,9	1,9	Gv	{691, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
8	11	34	1,1	4.321	16	16	Gv	{43, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
9	11	38	1,1	4.352	15	15	Gv	{4, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
10	12	41	1,1	4.545	31	31	Gv	{8, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
11	13	44	1,2	4.416	8,0	8,0	Gv	{2, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE

Usual Bricks Production (to produce all required nodes given default view parameters)

1/. First test

Drag a column header and drop it here to group by that column

				Duration (µs)				Grid Dimensions	Block Dimensions	Dynamic Shared Mem/Block	Stream ID	Occupancy	Registers Per Thread	Cache Configuration Executed	Shared Memory Configuration Executed
1	47	47	95	17.409	55	55	Gv	{1, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
2	58	10	95	15.682	40	40	Gv	{8, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
3	70	15	96	21.794	4,0	4,0	Gv	{64, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
4	82	21	96	71.686	15	15	Gv	{264, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
5	95	27	97	244.309	58	58	Gv	{1020, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
6	10	34	97	840.297	20	20	Gv	{3636, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
7	11	37	98	15.777	40	40	Gv	{8, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE

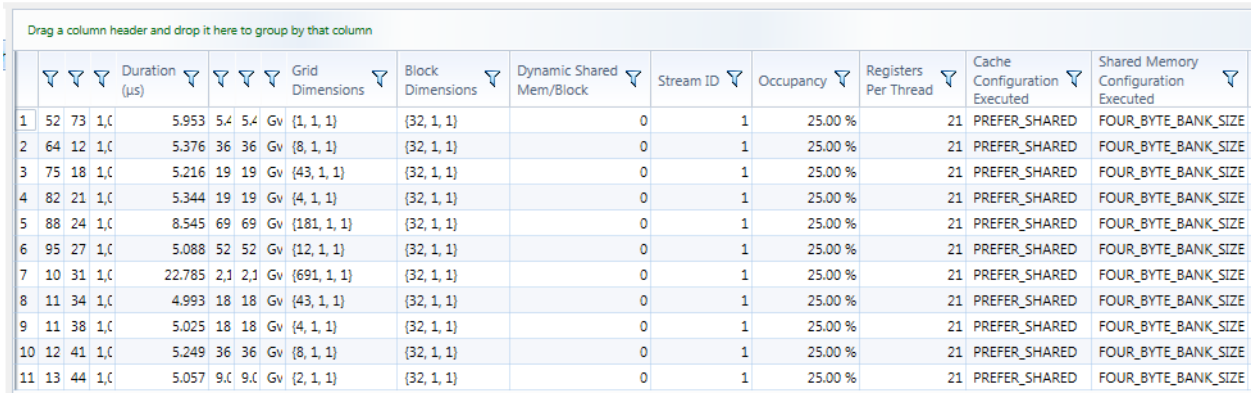
2/. Second test

Drag a column header and drop it here to group by that column

				Duration (µs)				Grid Dimensions	Block Dimensions	Dynamic Shared Mem/Block	Stream ID	Occupancy	Registers Per Thread	Cache Configuration Executed	Shared Memory Configuration Executed
1	47	47	1,1	17.410	65	65	Gv	{1, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
2	58	10	1,1	15.713	47	47	Gv	{8, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
3	70	15	1,1	18.690	2,8	2,8	Gv	{43, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
4	76	18	1,1	15.586	23	23	Gv	{4, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
5	83	21	1,1	53.765	12	12	Gv	{181, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
6	89	25	1,1	15.841	72	72	Gv	{12, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
7	96	28	1,1	177.039	45	45	Gv	{697, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
8	10	31	1,1	18.689	2,9	2,9	Gv	{45, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
9	11	35	1,1	646.777	17	17	Gv	{2667, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
10	11	38	1,1	21.250	4,1	4,1	Gv	{58, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
11	12	42	1,1	15.553	29	29	Gv	{5, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
12	13	45	1,2	15.554	29	29	Gv	{5, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
13	13	48	1,2	15.554	11	11	Gv	{2, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE

NODES => Remove 2 Synchronization Barriers :

1/. First test



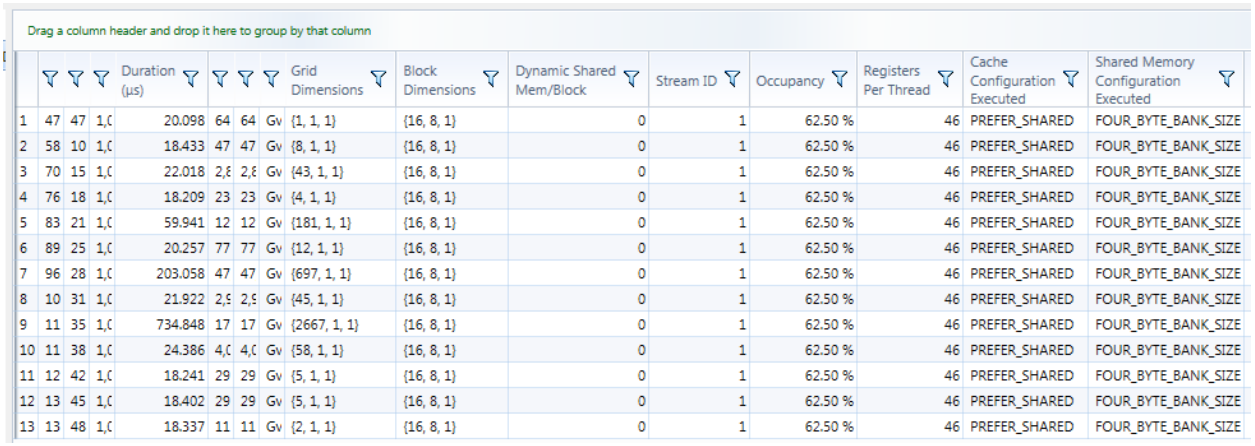
Drag a column header and drop it here to group by that column

				Duration (µs)				Grid Dimensions	Block Dimensions	Dynamic Shared Mem/Block	Stream ID	Occupancy	Registers Per Thread	Cache Configuration Executed	Shared Memory Configuration Executed
1	52	73	1,(	5.953	5,4	5,4	Gv	{1, 1, 1}	{32, 1, 1}	0	1	25.00 %	21	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
2	64	12	1,(	5.376	36	36	Gv	{8, 1, 1}	{32, 1, 1}	0	1	25.00 %	21	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
3	75	18	1,(	5.216	19	19	Gv	{43, 1, 1}	{32, 1, 1}	0	1	25.00 %	21	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
4	82	21	1,(	5.344	19	19	Gv	{4, 1, 1}	{32, 1, 1}	0	1	25.00 %	21	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
5	88	24	1,(	8.545	69	69	Gv	{181, 1, 1}	{32, 1, 1}	0	1	25.00 %	21	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
6	95	27	1,(	5.088	52	52	Gv	{12, 1, 1}	{32, 1, 1}	0	1	25.00 %	21	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
7	10	31	1,(	22.785	2,1	2,1	Gv	{691, 1, 1}	{32, 1, 1}	0	1	25.00 %	21	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
8	11	34	1,(	4.993	18	18	Gv	{43, 1, 1}	{32, 1, 1}	0	1	25.00 %	21	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
9	11	38	1,(	5.025	18	18	Gv	{4, 1, 1}	{32, 1, 1}	0	1	25.00 %	21	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
10	12	41	1,(	5.249	36	36	Gv	{8, 1, 1}	{32, 1, 1}	0	1	25.00 %	21	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
11	13	44	1,(	5.057	9,(	9,(	Gv	{2, 1, 1}	{32, 1, 1}	0	1	25.00 %	21	PREFER_SHARED	FOUR_BYTE_BANK_SIZE

2/. Second test

NODES and BRICKS => Remove 1 final Synchronization Barrier :

1/. First test



Drag a column header and drop it here to group by that column

				Duration (µs)				Grid Dimensions	Block Dimensions	Dynamic Shared Mem/Block	Stream ID	Occupancy	Registers Per Thread	Cache Configuration Executed	Shared Memory Configuration Executed
1	47	47	1,(	20.098	64	64	Gv	{1, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
2	58	10	1,(	18.433	47	47	Gv	{8, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
3	70	15	1,(	22.018	2,8	2,8	Gv	{43, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
4	76	18	1,(	18.209	23	23	Gv	{4, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
5	83	21	1,(	59.941	12	12	Gv	{181, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
6	89	25	1,(	20.257	77	77	Gv	{12, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
7	96	28	1,(	203.058	47	47	Gv	{697, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
8	10	31	1,(	21.922	2,9	2,9	Gv	{45, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
9	11	35	1,(	734.848	17	17	Gv	{2667, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
10	11	38	1,(	24.386	4,(	4,(	Gv	{58, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
11	12	42	1,(	18.241	29	29	Gv	{5, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
12	13	45	1,(	18.402	29	29	Gv	{5, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
13	13	48	1,(	18.337	11	11	Gv	{2, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE

2/. Second test

a/. NODES

Drag a column header and drop it here to group by that column

				Duration (µs)	Active Warp Time Self(µs)	Active Warp Time Total(µs)		Grid Dimensions	Block Dimensions	Dynamic Shared Mem/Block	Stream ID	Occupancy	Registers Per Thread	Cache Configuration Executed	Shared Memory Configuration Executed
1	52	73	1,(	5.921	5.472	5.472	Gv	{1, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
2	64	12	1,1	6.176	43.619	43.619	Gv	{8, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
3	75	18	1,1	5.665	212.562	212.562	Gv	{43, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
4	82	21	1,1	5.376	19.457	19.457	Gv	{4, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
5	88	24	1,1	8.928	740.608	740.608	Gv	{181, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
6	95	27	1,1	5.313	55.716	55.716	Gv	{12, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
7	10	31	1,1	23.714	2,226.688	2,226.688	Gv	{691, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
8	11	34	1,1	5.248	197.521	197.521	Gv	{43, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
9	11	38	1,1	5.184	18.753	18.753	Gv	{4, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
10	12	41	1,1	5.536	38.211	38.211	Gv	{8, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
11	13	44	1,1	5.345	9.728	9.728	Gv	{2, 1, 1}	{32, 1, 1}	0	1	25.00 %	17	PREFER_SHARED	FOUR_BYTE_BANK_SIZE

a/. BRICKS

Drag a column header and drop it here to group by that column

				Duration (µs)	Active Warp Time Self(µs)	Active Warp Time Total(µs)		Grid Dimensions	Block Dimensions	Dynamic Shared Mem/Block	Stream ID	Occupancy	Registers Per Thread	Cache Configuration Executed	Shared Memory Configuration Executed
1	47	47	1,(	20.162	64.613	64.613	Gv	{1, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
2	58	10	1,1	18.369	471.240	471.240	Gv	{8, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
3	70	15	1,1	22.082	2,815.315	2,815.315	Gv	{43, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
4	76	18	1,1	18.241	234.900	234.900	Gv	{4, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
5	83	21	1,1	61.893	13,032.808	13,032.808	Gv	{181, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
6	89	25	1,1	18.626	718.622	718.622	Gv	{12, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
7	96	28	1,1	203.730	47,417.033	47,417.033	Gv	{697, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
8	10	31	1,1	21.858	2,978.177	2,978.177	Gv	{45, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
9	11	35	1,1	733.951	179,433.103	179,433.103	Gv	{2667, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
10	11	38	1,1	25.090	4,099.618	4,099.618	Gv	{58, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
11	12	42	1,1	18.273	293.689	293.689	Gv	{5, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
12	13	45	1,1	18.210	293.241	293.241	Gv	{5, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE
13	13	48	1,1	18.338	118.090	118.090	Gv	{2, 1, 1}	{16, 8, 1}	0	1	62.50 %	46	PREFER_SHARED	FOUR_BYTE_BANK_SIZE

b. Force “Hardware Resources” at execution : launch_bounds()

…

c. CUDA “Cache Configurations” : L1 vs Shared Memory

…

d. Kernel “Launch Configurations” : grid/block sizes and thread access patterns

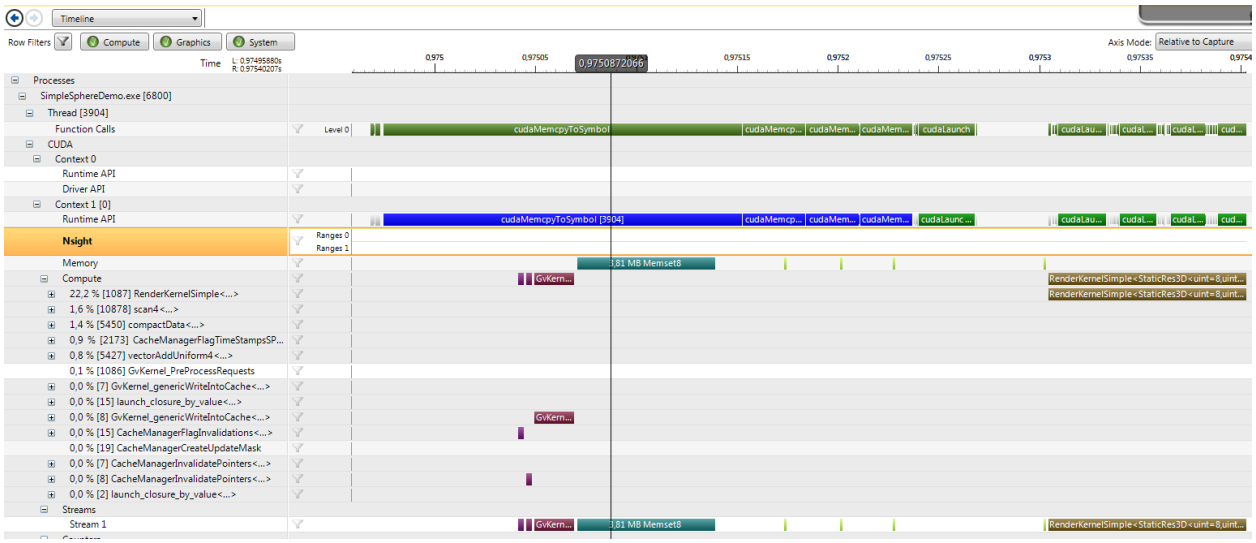
…

Other : Global “pipeline’s flow sequence” optimization

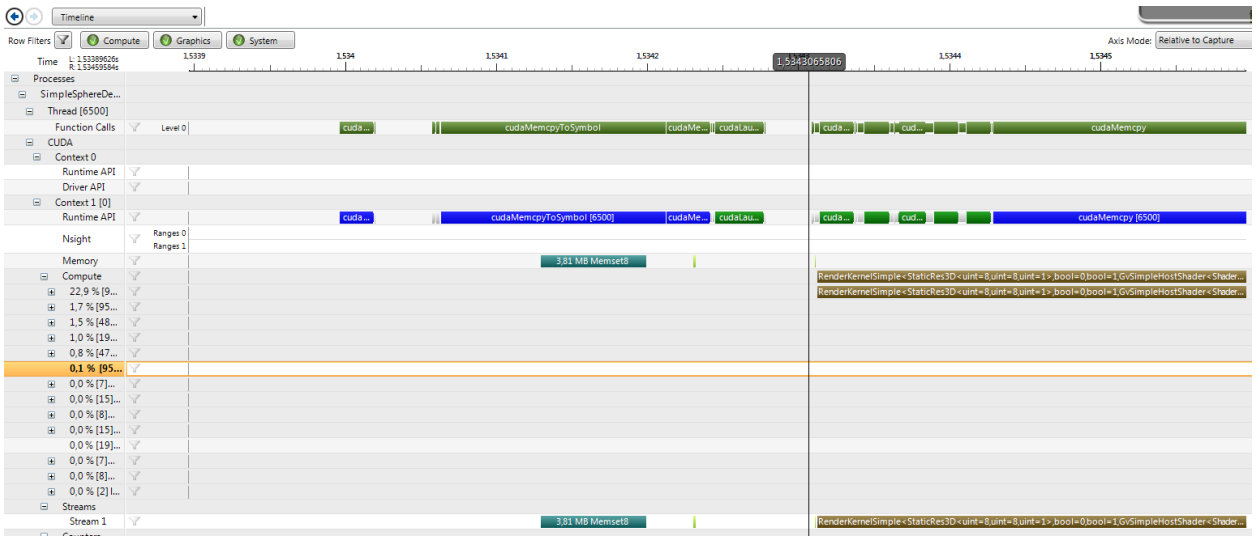
=> move that part in an other document

- remove/reduce number of calls to cudaMemcpyToSymbol() to update Constant Memory (i.e. equivalent of GLSL uniforms) to only modify values when required. Ex : “max_depth” of the octree should only change with a user-defined call in a GUI user interface, not at each render() call. As current graphics cards have only one 1 Copy Engine, copies are serialized.

Before : 4 calls to cudaMemcpyToSymbol()



After : only 2 calls to cudaMemcpyToSymbol()



4. Design & Architecture

TO DO
...

References

CUDA
- ...